HPC with GPUs

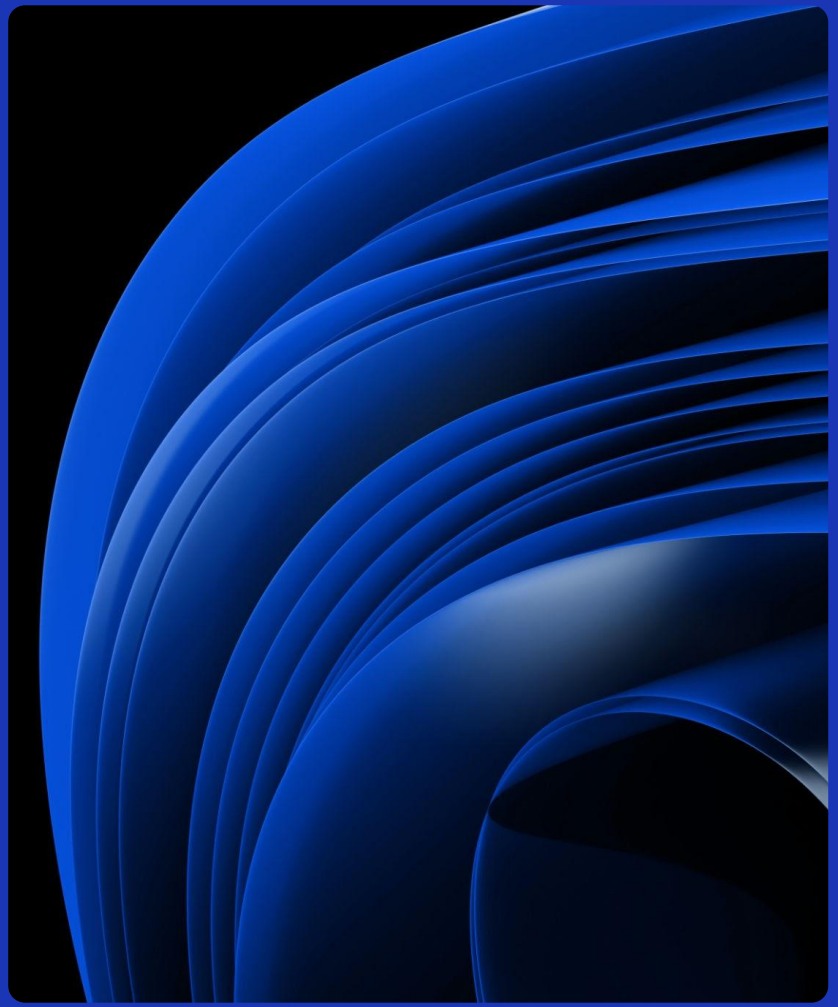
Neural Networks

Accelerated with GPUs

Muaz Ahmed

Arshaq Kirmani

Ibtehaj Haider



Confidential

Copyright ©

Intro

- What are Neural Networks?
- What was used to Train the NN?

Confidential

Copyright ©

Versions

Version 1

Serial

Processes each image through a fully connected network with ReLU and softmax layers, updating weights via backpropagation and gradient descent.

Version 2

Naive CUDA

Parallelizes basic operations like matrix multiplications across threads but processes each image sequentially, with limited GPU optimization and no shared memory usage.

Version 3

Optimized CUDA

Boosts performance by tuning launch configs, maximizing occupancy, minimizing communication, and efficiently using shared and constant memory.

Version 4

Tensor Cores

Builds on the optimized version by leveraging tensor cores for faster matrix operations, significantly accelerating training and inference.

Confidential

Copyright ©

Version 1

Properties, Stats and Results

Confidential

Copyright ©

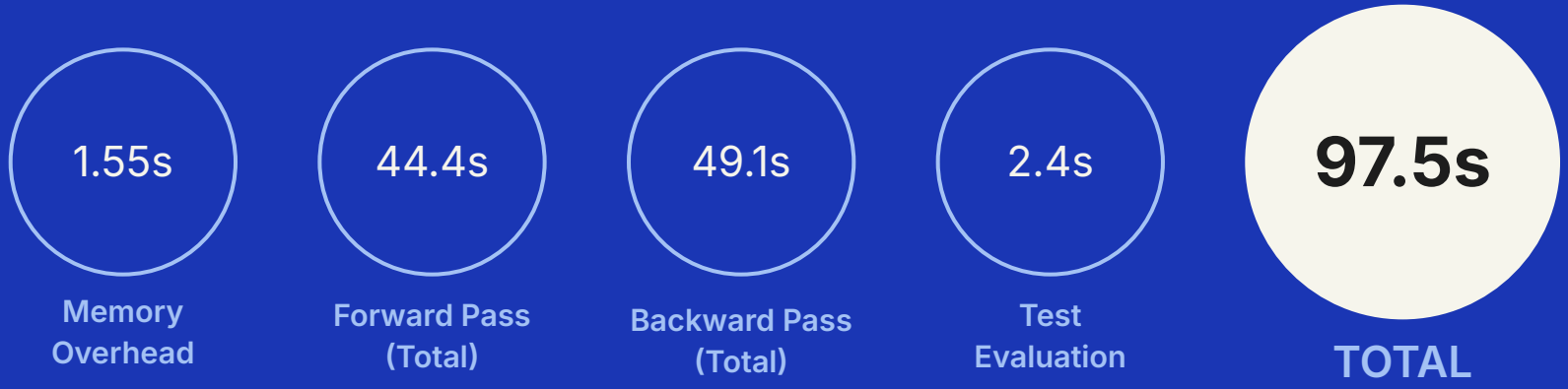
V1 Properties

- Runs on CPU without parallelism.
- Processes one image at a time.
- Uses a fully connected neural network with ReLU and softmax layers.
- Performs forward pass, loss calculation, and backpropagation sequentially.

Confidential

Copyright ©

Version 1 Time Divisions



Pure C-based implementation without any parallelization. Room for massive improvement in training phase.

Since there is little dependence at forward passes (different neurons training separately, it can easily be parallelized.

Confidential

Copyright ©

Version 2

Properties, Stats and Results

Confidential

Copyright ©

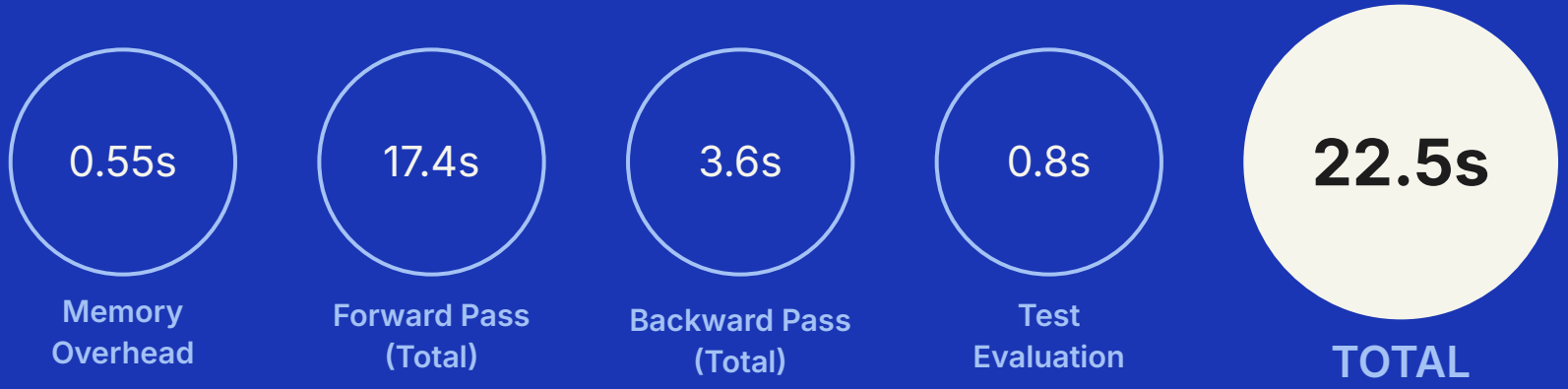
V2 Properties

- Runs on GPU with basic parallelism.
- Faster than CPU version but far from optimal performance.
- No shared memory or tensor cores used.
- Minimal optimizations in launch configuration.

Confidential

Copyright ©

Version 2 Time Divisions



A naive GPU Implementation. Still has some room for improvement by optimizing launch configurations, occupancy, communication and memory.

Confidential

Copyright ©

Version 3

Properties, Stats and Results

Confidential

Copyright ©

V3 Properties

- Fine-tuned launch configuration for balanced thread-block mapping.
- Maximized occupancy by optimizing register and shared memory usage.
- Reduced thread communication overhead through smarter data flow.
- Efficient use of shared and constant memory to minimize global memory access.
- Better performance and scalability than naive implementation.

Confidential

Copyright ©

Version 3 Time Divisions

The optimized CUDA implementation enhances performance by carefully tuning launch configurations, maximizing occupancy, reducing thread communication overhead, and leveraging the memory hierarchy with shared and constant memory for faster data access.

Confidential

Copyright ©

Version 4

Properties, Stats and Results

Confidential

Copyright ©

V4 Properties

- Utilizes tensor cores for improved matrix operations and higher throughput.
- Enhances parallelism with fine-tuned memory and launch configurations.
- Leverages memory hierarchy (shared, constant, and global memory) for faster data access.

Confidential

Copyright ©

Version 5

Properties, Stats and Results

Confidential

Copyright ©

V5 Properties

- Used #pragma acc for directive-based parallelism.
- Focused on developer productivity and code maintainability.

Confidential

Copyright ©

Version 5 Time Divisions

the OpenACC version, utilized #pragma acc for directive-based parallelism, prioritizing developer productivity and code maintainability. This implementation achieved an execution time of 19.98 seconds, with a speedup of 4.87x compared to the sequential version, and an accuracy of 92.53%.

Confidential

Copyright ©

Thank You

Questions?

Confidential

Copyright ©